Pablo Rauzy
pr@up8.edu
https://pablo.rauzy.name/teaching/mdlp

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Méthodologie de la programmation

Université Paris 8 – Vincennes à Saint-Denis
UFR MITSIC / L1 informatique

Séance 5 (TP) : MiniPaint avec Pygame

N'oubliez pas :
- Les TPs doivent être rendus par courriel au plus tard la veille de la séance suivante avec "[mdlp]" suivi du numéro de la séance et de votre nom dans le sujet du mail, par exemple "[mdlp] TP5 Rauzy".
- Quand un exercice demande des réponses qui ne sont pas du code, vous les mettrez dans un fichier texte `reponses.txt` à rendre avec le code.
- Le TP doit être rendu dans une archive, par exemple un tar gzippé obtenu avec la commande `tar czvf NOM.tgz NOM`, où `NOM` est le nom du répertoire dans lequel il y a votre code (idéalement, votre nom de famille et le numéro de la séance, par exemple "rauzy-tp5").
- Si l'archive est lourde (> 1 Mo), merci d'utiliser `https://bigfiles.univ-paris8.fr/`.
- Les fichiers temporaires (si il y en a) doivent être supprimés avant de créer l'archive.
- Le code doit être proprement indenté et les variables, fonctions, constantes, etc. correctement nommées, en respectant des conventions cohérentes.
- Le code est de préférence en anglais, les commentaires (si besoin) en français ou anglais, en restant cohérent.
- **N'hésitez jamais à chercher de la documentation par vous-même sur le net!**

Dans ce TP :
- Utilisation de la programmation orientée objet en Python.
  Documentation de Python : `https://docs.python.org/3/`.
- Premiers pas avec la bibliothèque Pygame.
  Documentation de Pygame : `http://pygame.org/docs/` (Attention, site moche!).

Exercice 0.

Récupération des fichiers nécessaires.

1. Pensez à organiser correctement votre espace de travail, par exemple tout ce qui se passe dans ce TP pourrait être dans `~/mdlp/s5-tp/`.
2. Récupérez les fichiers nécessaires depuis la page web du cours, ou directement en ligne de commande avec `wget https://pablo.rauzy.name/teaching/mdlp/seance-5_tp.tgz`.
3. Une fois que vous avez extrait le dossier de l'archive (par exemple avec `tar xzf seance-5_tp.tgz`), renommez le répertoire en votre nom (avec la commande `mv mdlp_seance-5_files` *`votre-nom`*). Si vous ne le faites pas tout de suite, pensez à le faire avant de rendre votre TP.
4. Ce TP va consister en la réalisation d'un petit logiciel de dessin simpliste en utilisant la bibliothèque Pygame. Le code qui vous est fourni fonctionne déjà, mais ne fait rien de vraiment intéressant.
   → Lancez le tout de même avec `python3 minipaint.py`.
   Attention, pour l'instant vous ne pouvez même pas fermer la fenêtre ! Et pour cause, aucun évènement utilisateur n'est géré, c'est vous qui allez implémenter tout ça. Pour fermer le programme, tuez le en revenant sur le terminal et en tapant `^C` (ctrl+c).
   Vous êtes encouragé à tester systématiquement votre logiciel après chaque modification de code, cela vous aidera à repérer les erreurs au plus tôt et donc le plus précisément possible.

Exercice 1.

État des lieux du code fourni.

1. Commençons avec le fichier principal : `minipaint.py`.
   (a) Le fichier commence par importer différents modules.
   → Que font les lignes 3 et 4 ?
   (b) Ensuite on définit une classe `MiniPaint`, qui est la classe principale de notre logiciel.
   → Quelles sont ses méthodes ? Où sont-elles utilisées ?
   (c) Dans son constructeur, on peut voir :
   - L'initialisation de Pygame.
   - La création d'une fenêtre.
   - La création d'une toile ("canvas" en anglais) pour dessiner qui permet d'utiliser la transparence (c'est à ça que servent les arguments donnés en plus après la taille).

— La définition d'une couleur de fond pour cette toile (les couleurs sont définies par 4 valeurs comprises entre 0 et 255 : rouge, vert, bleu, et alpha, qui définit l'opacité).
— L'initialisation de l'attribut `_shapes` qui va contenir les éléments du dessin (des formes géométriques, d'où le nom) que l'on crée.
Ceci mérite un peu plus d'explications :
Pygame fournit un mécanisme qui permet d'afficher efficacement un ensemble de sprites. Pour cela, la bibliothèque fournie une classe `pygame.sprite.Sprite` de laquelle on peut dériver pour définir un Sprite, et une classe `pygame.sprite.Group` qui sert de conteneur pour des instances de la classe `Sprite`. La classe `Group` a des méthodes `add` et `remove` qui prennent en argument un sprite, et une méthode `draw` qui prend en argument une surface sur laquelle dessiner tous les sprites d'un seul coup.
La classe `pygame.sprite.OrderedUpdates` est un conteneur de sprites qui dérive de la classe `Group` et qui a la spécificité de dessiner des sprites dans leur ordre d'ajout. C'est plus lent que de laisser Pygame faire ses optimisations internes, mais cela nous intéresse ici pour le dessin : on veut que la dernière forme dessinée apparaisse par dessus les précédentes.
— Ensuite, on a l'initialisation de deux autres attributs : `_palette` comme une instance de la classe `Palette` et `_tools` comme une instance de la classe `Tools`.
→ Modifiez le titre de la fenêtre pour y ajouter votre nom.

(d) Dans la méthode `run` on voit la boucle principale du programme et qu'il ne s'y passe pour l'instant pas grand chose.
→ Que s'y passe-t-il pour l'instant ?

2. Passons maintenant au fichier `palette.py`.
Comme vous le voyez il ne comporte pour l'instant pas grand chose. La classe `Palette` va nous servir à gérer la sélection de couleurs pour le dessin. On reviendra dessus plus tard.
3. Le fichier `tools.py` est plus intéressant. Essayez de lire son code et de le comprendre par vous même.
→ Dans la méthode `make_shape` que se passe-t-il au niveau du `return` ?
4. Le fichier `shape.py` implémente la classe `Shape`. Elle dérive de `pygame.sprite.Sprite` dont on a parlé à la question 1.1.c. Elle hérite donc de tous ses comportements, et on voit qu'elle ne redéfinit que son constructeur.
Dans celui-ci, elle appelle le constructeur parent, puis construit le sprite proprement dit à partir de la surface qui lui est donné en argument.
Un sprite est composé de deux choses : une image (`image`), et un rectangle (`rect`) qui indique la position et la taille du sprite.
En regardant la méthode `Tools.make_shape` on comprend ce qui se passe : lors de la création d'une forme géométrique, on la dessine d'abord sur une surface de la taille de la toile, mais ensuite on ne garde dans le sprite que la plus petite surface nécessaire pour contenir la figure [1], mais on se rappelle de sa position.
→ En vous aidant de la documentation de Pygame, expliquez l'appel à la méthode `blit` de la classe `Surface` présent dans le constructeur de la classe `Shape`.

Exercice 2.

Dans l'état où il vous est fournit, le code fonctionne et une fenêtre s'affiche, mais on ne peut pas encore dessiner. Pour cause aucun évènement utilisateur n'est géré.

Le but de cet exercice va être de mettre en place la gestion des évènements.

1. Retournons dans la méthode `MiniPaint.run`. On voudrait commencer par permettre de fermer le logiciel.
→ Dans la boucle de gestion des évènements, rajoutez un test pour voir si le type de l'évènement est `pygame.QUIT`, et dans ce cas faire en sorte d'arrêter la boucle principale.
2. On veut maintenant permettre à l'utilisateur/trice de dessiner.
Pour cela on veut que lorsqu'ille presse le bouton gauche de la souris sur la toile, on retienne la position du curseur comme origine de la forme à dessiner, et que lorsqu'ille relâche ce bouton, on dessine effectivement la forme en la créant et en l'ajoutant dans le conteneur de sprites à afficher.

(a) Le type de l'évènement "presse le bouton de la souris" est `pygame.MOUSEBUTTONDOWN`. Pour ce type d'évènement :
— L'attribut `button` de l'évènement vaut 1, 2, ou 3 pour respectivement le bouton gauche, milieu, ou droite de la souris.
— L'attribut `pos` de l'évènement vaut la position du curseur.
Dans la classe `Tools` (dont on rappelle qu'on a une instance dans l'attribut `_tools` de la classe `MiniPaint`), la méthode `set_position` sert à se rappeler de la position origine d'une figure.
→ Faites en sorte que lorsque l'utilisateur/trice presse le bouton gauche de sa souris sur la toile, on retienne la position du curseur comme origine de la forme à dessiner.

1. Cf. la documentation de Pygame : `https://pygame.org/docs/ref/surface.html#pygame.Surface.get_bounding_rect`.

(b) Le type de l'évènement "relâche le bouton de la souris" est `pygame.MOUSEBUTTONUP`. Il contient les mêmes attributs que `MOUSEBUTTONDOWN`.

Cette fois-ci, on veut appeler la méthode `make_shape` de `Tools` en lui passant en paramètre la couleur courante (qu'on récupère depuis la palette avec sa méthode `get_color`), et la position où le bouton a été relâché.

Le résultat de cette méthode doit être ajouté au groupe des sprites à afficher avec la méthode `add` de la classe `pygame.sprite.OrderedUpdates` (dont on rappelle qu'on a une instance dans l'attribut `_shapes` de la classe `MiniPaint`).

→ Écrivez le code de gestion de cet évènement.

3. On souhaite maintenant implémenter le changement d'outils. Dans notre interface minimaliste, cela va se faire à l'aide du clavier. L'idée est que quand on appuie sur la touche majuscule et une autre touche en même temps, cela correspond à un changement d'outil : majuscule + L pour les lignes, majuscule + C pour les cercles, majuscule + R pour les rectangles, et majuscule + P pour la palette de couleurs.

   Comme vous pouvez le voir, il y a déjà dans la classe `Tools` une méthode permettant de faire ce changement qui reçoit en argument une touche du clavier telle qu'encodée par Pygame (constante du type `pygame.K_*`).

   L'évènement Pygame correspondant à l'appui d'une touche sur le clavier est `pygame.KEYDOWN`. Pour ce type d'évènement :
   — L'attribut `key` contient la touche appuyée.
   — L'attribut `mod` contient un masque binaire représentant les touches modificatrices enfoncées.
   Un masque binaire permet de représenter l'état de plusieurs touches à la fois (ctrl, alt, maj…). En fait, il utilise un bit par touche, et on a juste besoin de regarder si le bit de la touche qui nous intéresse est à 1 (enfoncée) ou à 0 (pas). Pour cela on peut juste appliquer un "et logique", noté `&` en Python, avec la constante qui représente la touche : `pygame.KMOD_CTRL`, `pygame.KMOD_ALT`, `pygame.KMOD_SHIFT`, etc.
   Par exemple :

```python
if event.mod & pygame.KMOD_CTRL:
    if event.key == pygame.K_s:
        # code pour quand on a fait ctrl+s
```

   → Écrivez le code qui permet de changer d'outil.

4. Comme après chaque question, testez votre code.
   → Que remarquez-vous ?

Exercice 3.

Comme vous avez pu le constater à la fin de l'exercice précédent, les outils ne sont pas vraiment implémentés et on peut seulement faire des points noirs pour l'instant.

Le but de cet exercice est de faire marcher les trois outils de dessin : ligne, cercle, rectangle.

1. On va commencer par les lignes.

   Dans la méthode `make_shape` de la classe `Tools`, à la place du code qui dessine un cercle de rayon 1 (les petits points noirs que l'on peut dessiner pour l'instant), on veut mettre du code qui permet de dessiner une ligne qui part de là où l'on a pressé le bouton de la souris et va jusqu'à là où on le relâche.

   → À l'aide de la documentation du module `draw` de Pygame (`https://pygame.org/docs/ref/draw.html`), écrivez le code permettant de tracer des lignes.

2. Une fois que les lignes fonctionnent, on veut implémenter les deux autres outils.

   Attention, il va falloir calculer le rayon du cercle, pour cela vous aurez besoin de la fonction `sqrt` (pour "SQUARE ROOT", racine carrée) disponible dans le module `math` de la bibliothèque standard de Python.

   → Mettez à jour la méthode `make_shape` pour que la forme dessinée dépende de l'outil en cours d'utilisation (attribut `_active`).

3. Votre logiciel de dessin sait maintenant faire des lignes, des cercles et des rectangles ! Il suffit pour cela de changer d'outil avec les raccourcis clavier définis précédemment et d'utiliser la souris pour tracer l'élément. Par contre, les éléments ne sont affichés qu'une fois qu'on a relâché la souris, ce qui n'est pas très pratique. On voudrait que l'élément en cours de création soit affiché aussi.

   Pour cela on va devoir prendre en compte les mouvements de la souris. Le type de l'évènement "la souris bouge" est `pygame.MOUSEMOTION`. Pour ce type d'évènement :
   — L'attribut `buttons` de l'évènement est un triplet de valeurs qui valent 1 ou 0 suivant que le bouton correspondant est pressé ou pas.
   — L'attribut `pos` de l'évènement vaut la position du curseur.

   (a) → Dans la boucle d'évènements (dans la méthode `MiniPaint.run`), lors d'un mouvement de la souris, si le bouton gauche est enfoncé, faites appel à la méthode `make_shape`, mais plutôt que de mettre son résultat dans `_shapes` pour affichage définitif, stockez le simplement dans une variable temporaire (disons `tmp`) qui servira à l'affichage jusqu'à ce que le bouton de la souris soit relâché.

(b) En dessous du commentaire `draw current shape`, on veut afficher la forme en cours de dessin. Mais on ne veut rien faire si on est pas en train de dessiner. Il y a plusieurs méthodes valides pour faire cela, mais le plus simple est peut-être de s'arranger pour toujours avoir la valeur `None` dans `tmp` quand il n'y a pas une forme en train d'être dessinée.

→ Initialisez la variable `tmp` à `None` avant la boucle principale du programme et remettez là à `None` quand on a fini de dessiner (c'est à dire qu'on relâche le bouton de la souris).

(c) Pour afficher le sprite `tmp` sur une surface, on appelle la méthode `blit` de cette surface avec comme argument l'image du sprite (`tmp.image`) et sa position (`tmp.rect`).

→ En dessous du commentaire `draw current shape`, si on est en train de dessiner, alors mettez sur la toile (`_canvas`) le dessin en cours (`tmp`).

Exercice 4.

Dans cet exercice on va implémenter le choix de la couleur de dessin.

1. La stratégie que l'on va utiliser pour le choix des couleurs est d'afficher une image contenant plein de couleurs et de permettre à l'utilisateur/trice de choisir une couleur en cliquant dessus.

   Pour cela il nous faut donc charger en mémoire une telle image. Il y en a une, `color-picker.png` qui vous est fournie, mais vous pouvez en choisir une autre si vous préférez (ne perdez pas de temps pour ça cela dit).

   → Dans le constructeur de la classe `Palette`, chargez dans l'attribut `_palette` l'image `color-picker.png` à l'aide de la fonction `pygame.image.load` (voir la documentation).

2. On va maintenant construire une surface qui servira de voile par dessus notre dessin le temps d'afficher la palette de couleurs.

   (a) → Toujours dans le constructeur de `Palette`, définissez un nouvel attribut `_cp` (pour "color Picker") comme une surface de la taille de la toile (que l'on reçoit en paramètre dans le constructeur) avec les mêmes propriétés que `_canvas` dans `MiniPaint`.

   (b) → À l'aide de la méthode `fill`, remplissez la surface `_cp` d'un voile semi-opaque, par exemple avec la couleur `pygame.Color(0, 0, 0, 150)` (c'est à dire du noir, avec le dernier paramètre qui donne le niveau d'opacité).

   (c) → Maintenant, collez l'image de la palette par dessus le voile, à l'aide de la méthode `blit` et aux coordonnées (0, 0) car ce sera plus simple à traiter.

3. Maintenant on veut pouvoir afficher la palette de couleurs, et en sortir.

   (a) → Faites retourner la palette (`_cp`) à la méthode `get_color_picker` de `Palette`.

   (b) → Dans la boucle principale au niveau du commentaire `display color picker`, affichez la palette (c'est à dire ce que renvoie la méthode `get_color_picker` de l'attribut `_palette`) aux coordonnées (0,0) de la toile, si on est en train d'utiliser la palette (vous pouvez récupérer l'outil en cours d'utilisation avec la méthode `get_active_tool` de la classe `Tools`).

   (c) On voudrait aussi pouvoir sortir de la palette et revenir automatiquement à l'outil où on était avant d'appeler la palette en appuyant simplement sur la touche échap (`pygame.K_ESCAPE`).

   Bonus → Implémenter ce comportement en utilisant la méthode `switch_back` de `Tools`.

4. Maintenant, on veut faire marcher la palette pour pouvoir effectivement choisir une couleur.

   (a) On va commencer par implémenter la méthode `update_color`. Cette méthode reçoit une position en paramètre. Si celle-ci est au dessus de l'image de la palette de couleurs (qui fait 255 par 255 pixels), alors on veut récupérer la couleur de l'image là où se trouve le curseur, sinon, renvoyer du noir comme la méthode le fait pour l'instant.

   Pour récupérer la couleur d'un pixel à une position donnée dans une surface, on peut utiliser la méthode `get_at`.

   → Implémentez la méthode `update_color`.

   (b) → Dans la boucle de gestion des évènements, lorsque la souris bouge, si l'outil actif est la palette, alors appelez la méthode `update_color` de la palette en lui passant la position du curseur en paramètre.

   (c) → Quand l'utilisateur/trice clique, si l'outil actif est la palette, alors appelez la méthode `pick_color` pour mettre à jour la couleur.

Exercice 5.

Maintenant tout fonctionne, vous pouvez utiliser votre logiciel de dessin !
Mais il est encore possible de l'améliorer un peu...

1. On peut notamment améliorer la palette de couleurs, en affichant la couleur actuellement sélectionnée et celle qui est couramment survolée.

   → Dans la méthode `get_color_picker` de `Palette`, affichez un rectangle de couleur correspondant à la couleur sélectionnée et à la couleur survolée (ces rectangles doivent apparaître en dehors de la palette elle-même évidemment).

2. Maintenant, on aimerait bien pouvoir sauvegarder nos jolis dessins.

   → Quand l'utilisateur/trice fait `^S` (ctrl+s), sauvez l'image de la toile dans un fichier PNG en utilisant la fonction `pygame.image.save` (consultez la documentation).